Kapitel DB:V

V. Die relationale Datenbanksprache SQL

- Einführung
- SQL als Datenanfragesprache
- SQL als Datendefinitionssprache
- SQL als Datenmanipulationssprache
- Sichten
- SQL vom Programm aus

Einführung

Historie [Wikipedia] [modern-sql.com]

1974 SEQUEL. *Structured English Query Language* bildet Ausgangspunkt.

⋮ Wildwuchs verschiedener Dialekte.

1986 SQL-86 (SQL-87, SQL1). Als ANSI-Standard verabschiedet.

1989 SQL-89. Integritäts-Constraints.

1992 SQL-92 (SQL2). Umfangreiche Erweiterungen für Datentypen und Bindings.

1999 SQL:1999 (SQL3). Rekursive Anfragen.

2003 SQL:2003. Einführung von XML-Features.

2006 SQL:2006. Verwendung von SQL in Zusammenhang mit XML.

2011 SQL:2011. Temporale Daten.

2016 SQL:2016. Row-Pattern-Matching, polymorphe Tabellen, JSON.

2019 SQL:2019. Mehrdimensionale Felder.

2023 SQL:2023. Datentyp JSON, Graph-Query-Language GQL.

Bemerkungen:

- SEQUEL wurde bei IBM-Research, San Jose, entwickelt. Es diente als Schnittstelle zum experimentellen relationalen Datenbanksystem „R“.
- SQL ist das Acronym für *Structured Query Language*.

Einführung

Vergleich zu theoretischen Anfragesprachen

Relationen in SQL:

- sind im Allgemeinen nicht duplikatfrei sondern Multimengen
- Duplikatfreiheit in Basisrelationen wird mit Integritätsbedingungen realisiert; in Ergebnisrelationen müssen Duplikate explizit entfernt werden.

Anfragen in SQL:

- bilden die Relationenalgebra weitgehend ab
- besitzen Grenzen hinsichtlich der Orthogonalität
- enthalten zusätzlich Operationen zur Aggregierung, Gruppierung, Sortierung, Verarbeitung spezieller Datentypen

weitere Konzepte von SQL:

- Definition von Datenbanken
- Pflege und Modifikation von Relationen
- Verwaltung von Benutzern, Autorisierung

Einführung

Komponenten von SQL

1. Datenanfragesprache, DQL. (*Data Query Language*)
   Formulierung von Anfragen, Auswahl und Aufbereitung von Daten

2. Datendefinitionssprache, DDL. (*Data Definition Language*)
   Definition und Modifikation der Datenstrukturen für Datenbanken:
   - externe Ebene: Sichten und Zugriffsrechte (Autorisierung)
   - konzeptuelle Ebene: Relationenschemata und Integritätsbedingungen
   - physische Ebene: Art des Index, Zugriffsoptimierung

3. Datenmanipulationssprache, DML. (*Data Manipulation Language*)
   Einfügen, Ändern und Löschen von Daten

4. Datenkontrollsprache, DCL. (*Data Control Language*)
   Vergabe und Entzug von Berechtigungen

Einführung

Komponenten von SQL

1. Datenanfragesprache, DQL. (*Data Query Language*)
   Formulierung von Anfragen, Auswahl und Aufbereitung von Daten

2. Datendefinitionssprache, DDL. (*Data Definition Language*)
   Definition und Modifikation der Datenstrukturen für Datenbanken:
   - externe Ebene: Sichten und Zugriffsrechte (Autorisierung)
   - konzeptuelle Ebene: Relationenschemata und Integritätsbedingungen
   - physische Ebene: Art des Index, Zugriffsoptimierung

3. Datenmanipulationssprache, DML. (*Data Manipulation Language*)
   Einfügen, Ändern und Löschen von Daten

4. Datenkontrollsprache, DCL. (*Data Control Language*)
   Vergabe und Entzug von Berechtigungen

Bemerkungen:

- Die SQL Datenanfragesprache kompakt:
  1. Illustration der Grundideen
  2. Syntax des Select-From-Where-Blocks
  3. Select- / From- / Where-Klausel im Detail
  4. Bezug zur Relationenalgebra und zum Tupelkalkül
  5. Geschachtelte Anfragen (aka Unterabfragen bzw. Subqueries)
  6. Allquantifizierung
  7. Mengenoperationen
  8. Aggregat- und Gruppierungsfunktionen
  9. Nullwerte
  10. Zusammengesetzte Terme
  11. SQL-89 versus SQL 92

Kapitel DB:V

V. Die relationale Datenbanksprache SQL

- Einführung
- SQL als Datenanfragesprache
- SQL als Datendefinitionssprache
- SQL als Datenmanipulationssprache
- Sichten
- SQL vom Programm aus

SQL als Datenanfragesprache

Illustration der Grundideen

Kern von SQL-Anfragen ist der Select-From-Where-Block (SFW-Block) :

- `select`
  Spezifiziert die Attribute des Ergebnisschemas.
- `from`
  Spezifiziert die verwendeten Relationen (Basis oder abgeleitete).
- `where`
  Spezifiziert Selektions- und Verbundbedingungen.

SQL als Datenanfragesprache

Illustration der Grundideen

Kern von SQL-Anfragen ist der Select-From-Where-Block (SFW-Block) :

- `select`
  Spezifiziert die Attribute des Ergebnisschemas.
- `from`
  Spezifiziert die verwendeten Relationen (Basis oder abgeleitete).
- `where`
  Spezifiziert Selektions- und Verbundbedingungen.

- `group by`
  Spezifiziert die Attribute, hinsichtlich derer Tupel gruppiert werden.
- `having`
  Spezifiziert Selektionsbedingung für Gruppen.

SQL als Datenanfragesprache

Illustration der Grundideen

Kern von SQL-Anfragen ist der Select-From-Where-Block (SFW-Block) :

- `select`
  Spezifiziert die Attribute des Ergebnisschemas.
- `from`
  Spezifiziert die verwendeten Relationen (Basis oder abgeleitete).
- `where`
  Spezifiziert Selektions- und Verbundbedingungen.
- `group by`
  Spezifiziert die Attribute, hinsichtlich derer Tupel gruppiert werden.
- `having`
  Spezifiziert Selektionsbedingung für Gruppen.
- `order by`
  Spezifiziert Prädikate zur Sortierung der Ergebnistupel.
- `union`
  Ermöglicht Vereinigung mit Ergebnistupeln nachfolgender SFW-Blöcke.

SQL als Datenanfragesprache

Illustration der Grundideen [SFW-Block Intro]

Buch		
ISBN	Titel	Verlag
1-2033-1113-8	Brecht Lesebuch	Piper
3-8244-2012-X	Laengengrad	Berlin-Verlag
0-8053-1753-8	Fundamentals of Database Systems	Addison Wesley
0-2314-2305-X	Heuristics	Addison Wesley

```
select *
from Buch
```
[SQLPad]

$\rightsquigarrow$

ISBN	Titel	Verlag
1-2033-1113-8	Brecht Lesebuch	Piper
3-8244-2012-X	Laengengrad	Berlin-Verlag
0-8053-1753-8	Fundamentals of Database Systems	Addison Wesley
0-2314-2305-X	Heuristics	Addison Wesley

SQL als Datenanfragesprache

Illustration der Grundideen [SFW-Block Intro]

Buch		
ISBN	Titel	Verlag
1-2033-1113-8	Brecht Lesebuch	Piper
3-8244-2012-X	Laengengrad	Berlin-Verlag
0-8053-1753-8	Fundamentals of Database Systems	Addison Wesley
0-2314-2305-X	Heuristics	Addison Wesley

```
select Titel, Verlag
from Buch
```
[SQLPad]

$\rightsquigarrow$

Titel	Verlag
Brecht Lesebuch	Piper
Laengengrad	Berlin-Verlag
Fundamentals of Database Systems	Addison Wesley
Heuristics	Addison Wesley

SQL als Datenanfragesprache

Illustration der Grundideen [SFW-Block Intro]

Buch		
ISBN	Titel	Verlag
1-2033-1113-8	Brecht Lesebuch	Piper
3-8244-2012-X	Laengengrad	Berlin-Verlag
0-8053-1753-8	Fundamentals of Database Systems	Addison Wesley
0-2314-2305-X	Heuristics	Addison Wesley

```
select Titel, Verlag
from Buch
where Verlag = 'Addison Wesley'
```
[SQLPad]

$\rightsquigarrow$

Titel	Verlag
Fundamentals of Database Systems	Addison Wesley
Heuristics	Addison Wesley

SQL als Datenanfragesprache

Illustration der Grundideen [SFW-Block Intro]

Buch		
ISBN	Titel	Verlag
1-2033-1113-8	Brecht Lesebuch	Piper
3-8244-2012-X	Laengengrad	Berlin-Verlag
0-8053-1753-8	Fundamentals of Database Systems	Addison Wesley
0-2314-2305-X	Heuristics	Addison Wesley

```
select Verlag
from Buch
```
[SQLPad]

$\rightsquigarrow$

Verlag
Piper
Berlin-Verlag
Addison Wesley
Addison Wesley

SQL als Datenanfragesprache

Illustration der Grundideen [SFW-Block Intro]

Buch		
ISBN	Titel	Verlag
1-2033-1113-8	Brecht Lesebuch	Piper
3-8244-2012-X	Laengengrad	Berlin-Verlag
0-8053-1753-8	Fundamentals of Database Systems	Addison Wesley
0-2314-2305-X	Heuristics	Addison Wesley

```
select distinct Verlag
from Buch
```
[SQLPad]

$\rightsquigarrow$

Verlag
Piper
Berlin-Verlag
Addison Wesley

SQL als Datenanfragesprache

Syntax des Select-From-Where-Blocks

```
select [all | distinct]
   {*| <attribute1> [[as] <alias1>], <attribute2> [[as] <alias2>], ...}

from <table1> [[as] <alias1>], <table2> [[as] <alias2>], ...

[where <condition>]

[group by <attribute1>, <attribute2>, ...]
[having <condition>]

[order by <attribute1>, <attribute2>, ...[asc | desc]]
[union [all]]
[limit [<offset_num>,] <tuple_num>]
```

Bemerkungen:

- Die Notation der Syntax ist an der Backus-Naur-Form (BNF) angelehnt.
- Die dargestellte Syntax enthält die wichtigsten Elemente. Moderne Datenbanksysteme stellen noch eine Reihe von Erweiterungen zur Verfügung, die über das hier notierte Schema hinausgehen. Siehe beispielsweise das `select`-Statement bei MySQL.
- `[[as] <alias>]` deklariert zusätzliche Bezeichner für Attribute und Tupelvariablen im lexikalischen Gültigkeitsbereich des SFW-Blocks.
- `<condition>` ist eine Formel, die aus Atomen und logischen Junktoren aufgebaut ist. Die Atome entsprechen weitgehend den Atomen im Tupel- und Domänenkalkül.
- Seit SQL-92 sind in der From-Klausel auch Join-Operatoren oder ein SFW-Block zugelassen, um eine neue (abgeleitete) Relation aufzubauen.

SQL als Datenanfragesprache

From-Klausel [SFW-Block-Syntax]

```
from <table1> [[as] <alias1>], <table2> [[as] <alias2>], ...
```

- Die From-Klausel spezifiziert die Relationen einer Anfrage und bildet somit den *Ausgangspunkt* für die Anfragebearbeitung.
- Eine Komma-separierte Liste von Relationen entspricht der Bildung des kartesischen Produktes.
- Die Verwendung von Aliassen entspricht der Einführung von *Tupelvariablen*, die zur Qualifizierung von Attributen verwendet werden können.
- Aliasse ermöglichen die mehrfache Spezifikation desselben Attributes einer Relation zur Formulierung tupelübergreifender Bedingungen.
  Stichwort: Selbstverbund (*Self-Join*)

SQL als Datenanfragesprache

From-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

```
select * 
from Mitarbeiter as m, Mitarbeiter as employee
```
[SQLPad (flawed)]

$\rightsquigarrow$ Bildung des kartesischen Produktes,
Ausgabe einer Tabelle mit 10 Spalten und 16 Zeilen

SQL als Datenanfragesprache

Select-Klausel [SFW-Block-Syntax]

```
select [all | distinct]
    {*| <attribute1> [[as] <alias1>], <attribute2> [[as] <alias2>], ...}
```

- Die Select-Klausel spezifiziert die Attribute A_i des Ergebnisschemas. Die A_i müssen aus den in der From-Klausel spezifizierten Relationen r_j stammen. Mittels „$*$“ (*Wildcard*) werden alle Attribute ausgewählt.
- Zur Unterscheidung gleichbenannter Attribute A in verschiedenen Relationen r_1, r_2 ist eine Qualifizierung mittels Tupelvariablen möglich: $r_1.A$, $r_2.A$.
  Für jede Basisrelation r ist implizit eine Tupelvariable mit dem Namen der Relation vereinbart.
- Die Verwendung von Aliassen bedeutet eine Umbennung von Attributen im Ergebnisschema.
- Das Schlüsselwort `distinct` gibt an, ob die Tupel der Ergebnisrelation eine Menge oder eine Multimenge bilden.

Bemerkungen:

- Die mittels `distinct` erzwungene Duplikateliminierung ist nicht der Default. Gründe:
  - Duplikateliminierung erfordert in der Regel eine (aufwändige) Sortierung.
  - Bei Anfragen, die alle Tupel betreffen, kann die Eliminierung von Duplikaten zur Ergebnisverfälschung führen.

    Beispiel: Berechnung der Summe von Gehältern. Falls zwei Gehälter gleich sind, würde eines wegfallen.

SQL als Datenanfragesprache

Select-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

Folgende Anfragen sind äquivalent: [SQLPad]

```
select PersNr
from Mitarbeiter
```

```
select Mitarbeiter.PersNr
from Mitarbeiter
```

```
select m.PersNr
from Mitarbeiter m
```

```
select m.PersNr
from Mitarbeiter as m
```

SQL als Datenanfragesprache

Select-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

Folgende Anfragen sind äquivalent: [SQLPad]

```
select PersNr
from Mitarbeiter
```

```
select Mitarbeiter.PersNr
from Mitarbeiter
```

```
select m.PersNr
from Mitarbeiter m
```

```
select m.PersNr
from Mitarbeiter as m
```

SQL als Datenanfragesprache

Select-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

Folgende Anfragen sind äquivalent: [SQLPad]

```
select PersNr
from Mitarbeiter
```

```
select Mitarbeiter.PersNr
from Mitarbeiter
```

```
select m.PersNr
from Mitarbeiter m
```

```
select m.PersNr
from Mitarbeiter as m
```

SQL als Datenanfragesprache

Select-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

Folgende Anfragen sind äquivalent: [SQLPad]

```
select PersNr
from Mitarbeiter
```

```
select Mitarbeiter.PersNr
from Mitarbeiter
```

```
select m.PersNr
from Mitarbeiter m
```

```
select m.PersNr
from Mitarbeiter as m
```

Unerlaubte Anfrage: (wegen Mehrdeutigkeit von `PersNr`)

```
select PersNr
from Mitarbeiter, ArbeitetInProjekt
```

SQL als Datenanfragesprache

Where-Klausel [SFW-Block-Syntax]

`[where <condition>]`

- Die Where-Klausel dient zur Selektion von Tupeln aus den Relationen, die in der From-Klausel spezifiziert sind. Alle Tupel, die `<condition>` erfüllen, werden in die Ergebnismenge aufgenommen.
- `<condition>` entspricht einer logischen Formel, vergleichbar der Formel α im Tupelkalkül oder Domänenkalkül. Ausnahmen bzw. Ergänzungen sind u.a.:
  - die Junktoren heißen `and`, `or`, `not`
  - es gibt mengenwertige Operanden; die Operatoren hierfür sind `in`, `exists`, `any`
  - der Allquantor ist nicht zugelassen
  - ein Operand kann eine Unterabfrage, also wiederum ein komplexer SFW-Block, sein

SQL als Datenanfragesprache

Where-Klausel [SFW-Block-Syntax]

`[where <condition>]`

- Die Where-Klausel dient zur Selektion von Tupeln aus den Relationen, die in der From-Klausel spezifiziert sind. Alle Tupel, die `<condition>` erfüllen, werden in die Ergebnismenge aufgenommen.
- `<condition>` entspricht einer logischen Formel, vergleichbar der Formel α im Tupelkalkül oder Domänenkalkül. Ausnahmen bzw. Ergänzungen sind u.a.:
  - die Junktoren heißen `and`, `or`, `not`
  - es gibt mengenwertige Operanden; die Operatoren hierfür sind `in`, `exists`, `any`
  - der Allquantor ist nicht zugelassen
  - ein Operand kann eine Unterabfrage, also wiederum ein komplexer SFW-Block, sein
- Der „=“-Operator realisiert einen (*Theta-*)*Join*, wenn er auf je ein Attribut zweier Relationen angewendet wird. Mit mehreren, durch `and` verknüpften Gleichheitsbedingungen lassen sich Mehrwege-Joins spezifizieren.
- Es existieren weitere Operatoren zur Bereichsselektion (`between`) und zum Mustervergleich (`like`).

SQL als Datenanfragesprache

Where-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

```
select *  [SQLPad]
from Mitarbeiter
where ChefPersNr < 8000
```

$\rightsquigarrow$

Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5

SQL als Datenanfragesprache

Where-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

```
select *  [SQLPad]
from Mitarbeiter
where ChefPersNr < 8000
```

$\rightsquigarrow$

Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5

```
select *
from Mitarbeiter
where Wohnort like '%oel%'
```

$\rightsquigarrow$

Name	PersNr	Wohnort	ChefPersNr	AbtNr
Wong	3334	Koeln	8886	5

SQL als Datenanfragesprache

Where-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

```
select *   [SQLPad (flawed)]
from Mitarbeiter as m, ArbeitetInProjekt as a
where m.PersNr = a.PersNr or ChefPersNr = 9876
```

SQL als Datenanfragesprache

Where-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

```
select *   [SQLPad (flawed)]
from Mitarbeiter as m, ArbeitetInProjekt as a
where m.PersNr = a.PersNr or ChefPersNr = 9876
```

	Name	PersNr	Wohnort	ChefPersNr	AbtNr	PersNr	ProjektNr
	Smith	1234	Weimar	3334	5	1234	1
	Smith	1234	Weimar	3334	5	1234	2
$\rightsquigarrow$	Zelaya	9998	Erfurt	9876	4	1234	1
	Zelaya	9998	Erfurt	9876	4	1234	2
	Zelaya	9998	Erfurt	9876	4	6668	3
	Zelaya	9998	Erfurt	9876	4	4534	1

SQL als Datenanfragesprache

Where-Klausel

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

```
select *   [SQLPad (flawed)]
from Mitarbeiter as m, ArbeitetInProjekt as a
where m.PersNr = a.PersNr or ChefPersNr = 9876
```

	Name	PersNr	Wohnort	ChefPersNr	AbtNr	PersNr	ProjektNr
	Smith	1234	Weimar	3334	5	1234	1
	Smith	1234	Weimar	3334	5	1234	2
$\rightsquigarrow$	Zelaya	9998	Erfurt	9876	4	1234	1
	Zelaya	9998	Erfurt	9876	4	1234	2
	Zelaya	9998	Erfurt	9876	4	6668	3
	Zelaya	9998	Erfurt	9876	4	4534	1

SQL als Datenanfragesprache

Where-Klausel: Self-Join

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

Anfrage

„Wer arbeitet in derselben Abteilung wie Smith?“

SQL als Datenanfragesprache

Where-Klausel: Self-Join

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

Anfrage

„Wer arbeitet in derselben Abteilung wie Smith?“

```
select m1.Name  [SQLPad]
from Mitarbeiter as m1, Mitarbeiter as m2
where m2.Name = 'Smith' and
      m1.AbtNr = m2.AbtNr and
      m1.Name != 'Smith';
```

SQL als Datenanfragesprache

Where-Klausel: Self-Join

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

Anfrage

„Wer arbeitet in derselben Abteilung wie Smith?“

```
select m1.Name
from Mitarbeiter as m1, Mitarbeiter as m2
where m2.Name = 'Smith' and
      m1.AbtNr = m2.AbtNr and
      m1.Name != 'Smith';
```

[SQLPad]

Variante mit Subquery in From-Klausel [Subquery in Where-Klausel]

```
select m1.Name
from Mitarbeiter as m1,
     (select * from Mitarbeiter where Name = 'Smith') as m2
where m1.AbtNr = m2.AbtNr and
      m1.Name != 'Smith';
```

SQL als Datenanfragesprache

Where-Klausel: Self-Join

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

ArbeitetInProjekt	
PersNr	ProjektNr
1234	1
1234	2
6668	3
4534	1

Anfrage

„Wer arbeitet in derselben Abteilung wie Smith?“

```
select m1.Name
from Mitarbeiter as m1, Mitarbeiter as m2
where m2.Name = 'Smith' and
      m1.AbtNr = m2.AbtNr and
      m1.Name != 'Smith';
```

[SQLPad]

Variante mit Subquery in From-Klausel [Subquery in Where-Klausel]

```
select m1.Name
from (select * from Mitarbeiter where Name != 'Smith') as m1,
     (select * from Mitarbeiter where Name = 'Smith') as m2
where m1.AbtNr = m2.AbtNr;
```

SQL als Datenanfragesprache

Bezug zur Relationenalgebra

Seien r_1, r_2 Relationen über den Schemata $\mathcal{R}_1 = \{A_1, A_2\}$ bzw. $\mathcal{R}_2 = \{A_2, A_3\}$.

```
select A1, A3
from r1 as r3, r2
where r3.A2 = r2.A2
```

Äquivalenter Ausdruck in der Relationenalgebra:

$$\pi_{A_1,A_3}\sigma_{r_3.A_2=r_2.A_2}((\rho_{r_3}(r_1))\times r_2)$$

- SQL-Select entspricht der Projektion π
- SQL-From entspricht dem kartesischen Produkt $\times$
- SQL-Where entspricht der Selektion σ
- SQL-Alias-Deklaration entspricht der Umbennung ρ

SQL als Datenanfragesprache

Bezug zur Relationenalgebra

Seien r_1, r_2 Relationen über den Schemata $\mathcal{R}_1 = \{A_1, A_2\}$ bzw. $\mathcal{R}_2 = \{A_2, A_3\}$.

```
select A1, A3
from r1 as r3, r2
where r3.A2 = r2.A2
```

Äquivalenter Ausdruck in der Relationenalgebra:

$$\pi_{A_1,A_3}\sigma_{r_3.A_2=r_2.A_2}((\rho_{r_3}(r_1))\times r_2)$$

- SQL-Select entspricht der Projektion π
- SQL-From entspricht dem kartesischen Produkt $\times$
- SQL-Where entspricht der Selektion σ
- SQL-Alias-Deklaration entspricht der Umbennung ρ

SQL als Datenanfragesprache

Bezug zur Relationenalgebra

Seien r_1, r_2 Relationen über den Schemata $\mathcal{R}_1 = \{A_1, A_2\}$ bzw. $\mathcal{R}_2 = \{A_2, A_3\}$.

```
select A1, A3
from r1 as r3, r2
where r3.A2 = r2.A2
```

Äquivalenter Ausdruck in der Relationenalgebra:

$$\pi_{A_1,A_3}\sigma_{r_3.A_2=r_2.A_2}((\rho_{r_3}(r_1))\times r_2)$$

- SQL-Select entspricht der Projektion π
- SQL-From entspricht dem kartesischen Produkt $\times$
- SQL-Where entspricht der Selektion σ
- SQL-Alias-Deklaration entspricht der Umbennung ρ

SQL als Datenanfragesprache

Bezug zur Relationenalgebra

Seien r_1, r_2 Relationen über den Schemata $\mathcal{R}_1 = \{A_1, A_2\}$ bzw. $\mathcal{R}_2 = \{A_2, A_3\}$.

```
select A1, A3
from r1 as r3, r2
where r3.A2 = r2.A2
```

Äquivalenter Ausdruck in der Relationenalgebra:

$$\pi_{A_1,A_3}\sigma_{r_3.A_2=r_2.A_2}((\rho_{r_3}(r_1))\times r_2)$$

- SQL-Select entspricht der Projektion π
- SQL-From entspricht dem kartesischen Produkt $\times$
- SQL-Where entspricht der Selektion σ
- SQL-Alias-Deklaration entspricht der Umbennung ρ

SQL als Datenanfragesprache

Bezug zur Relationenalgebra

Seien r_1, r_2 Relationen über den Schemata $\mathcal{R}_1 = \{A_1, A_2\}$ bzw. $\mathcal{R}_2 = \{A_2, A_3\}$.

```
select A1, A3
from r1 as r3, r2
where r3.A2 = r2.A2
```

Äquivalenter Ausdruck in der Relationenalgebra:

$$\pi_{A_1,A_3}\sigma_{r_3.A_2=r_2.A_2}\big((\rho_{r_3}(r_1))\times r_2\big)$$

- SQL-Select entspricht der Projektion π
- SQL-From entspricht dem kartesischen Produkt $\times$
- SQL-Where entspricht der Selektion σ
- SQL-Alias-Deklaration entspricht der Umbennung ρ

SQL als Datenanfragesprache

Bezug zum Tupelkalkül

Seien r_1, r_2 Relationen über den Schemata $\mathcal{R}_1 = \{A_1, A_2\}$ bzw. $\mathcal{R}_2 = \{A_2, A_3\}$.

```
select A1, A3
from r1 as r3, r2
where r3.A2 = r2.A2
```

SQL als Datenanfragesprache

Bezug zum Tupelkalkül

Seien r_1, r_2 Relationen über den Schemata $\mathcal{R}_1 = \{A_1, A_2\}$ bzw. $\mathcal{R}_2 = \{A_2, A_3\}$.

```
select A1, A3
from r1 as r3, r2
where r3.A2 = r2.A2
```

- SQL-Select entspricht der Tupelsynthese mittels freier Variablen:

$$\{(t_3.A_1, t_2.A_3) \mid r_1(t_3) \ \wedge \ r_2(t_2) \ \wedge \ t_3.A_2 = t_2.A_2\}$$

SQL als Datenanfragesprache

Bezug zum Tupelkalkül

Seien r_1, r_2 Relationen über den Schemata $\mathcal{R}_1 = \{A_1, A_2\}$ bzw. $\mathcal{R}_2 = \{A_2, A_3\}$.

```
select A1, A3
from r1 as r3, r2
where r3.A2 = r2.A2
```

- SQL-Select entspricht der Tupelsynthese mittels freier Variablen:

$$\{(t_3.A_1, t_2.A_3) \mid r_1(t_3) \wedge r_2(t_2) \wedge t_3.A_2 = t_2.A_2\}$$

- SQL-From entspricht der Bindung von freien Variablen an Relationen:

$$\{(t_3.A_1, t_2.A_3) \mid r_1(t_3) \wedge r_2(t_2) \wedge t_3.A_2 = t_2.A_2\}$$

SQL als Datenanfragesprache

Bezug zum Tupelkalkül

Seien r_1, r_2 Relationen über den Schemata $\mathcal{R}_1 = \{A_1, A_2\}$ bzw. $\mathcal{R}_2 = \{A_2, A_3\}$.

```
select A1, A3
from r1 as r3, r2
where r3.A2 = r2.A2
```

- SQL-Select entspricht der Tupelsynthese mittels freier Variablen:

$$\{(t_3.A_1, t_2.A_3) \mid r_1(t_3) \wedge r_2(t_2) \wedge t_3.A_2 = t_2.A_2\}$$

- SQL-From entspricht der Bindung von freien Variablen an Relationen:

$$\{(t_3.A_1, t_2.A_3) \mid r_1(t_3) \wedge r_2(t_2) \wedge t_3.A_2 = t_2.A_2\}$$

- SQL-Where entspricht einem als Formel spezifiziertem Constraint:

$$\{(t_3.A_1, t_2.A_3) \mid r_1(t_3) \wedge r_2(t_2) \wedge t_3.A_2 = t_2.A_2\}$$

SQL als Datenanfragesprache

Geschachtelte Anfragen

Wichtige Verwendungsformen für eine Subquery in der Where-Klausel:

1. **select** $A_1, A_2, \ldots, A_n$
   from $r_1, r_2, \ldots, r_m$
   where [not] exists
   (**select... from... where...**)

   { Test auf leere Menge

SQL als Datenanfragesprache

Geschachtelte Anfragen

Wichtige Verwendungsformen für eine Subquery in der Where-Klausel:

1. **select** $A_1, A_2, \ldots, A_n$
   from $r_1, r_2, \ldots, r_m$
   where [not] exists
   (**select**... **from**... **where**...)

   { Test auf leere Menge

Beispiel: [Subquery in From-Klausel]

```
select m1.Name
from Mitarbeiter as m1
where exists
      (select *
       from Mitarbeiter as m2
       where m2.Name = 'Smith' and
             m2.AbtNr = m1.AbtNr and
             m1.Name != 'Smith');
```

SQL als Datenanfragesprache

Geschachtelte Anfragen

Wichtige Verwendungsformen für eine Subquery in der Where-Klausel:

1. **select** $A_1, A_2, \ldots, A_n$
   from $r_1, r_2, \ldots, r_m$
   where [not] exists
   (**select... from... where...**)

   { Test auf leere Menge

2. **select** $A_1, A_2, \ldots, A_n$
   from $r_1, r_2, \ldots, r_m$
   where $\{r_i.A_k \mid (r_i.A_k, r_j.A_l, \ldots)\}$ [not] in
   (**select... from... where...**)

   { Test auf Element in Menge

SQL als Datenanfragesprache

Geschachtelte Anfragen

Wichtige Verwendungsformen für eine Subquery in der Where-Klausel:

1. **select** $A_1, A_2, \ldots, A_n$
   from $r_1, r_2, \ldots, r_m$
   where [not] exists
   (**select... from... where...**)

   { Test auf leere Menge

2. **select** $A_1, A_2, \ldots, A_n$
   from $r_1, r_2, \ldots, r_m$
   where $\{r_i.A_k \mid (r_i.A_k, r_j.A_l, \ldots)\}$ [not] in
   (**select... from... where...**)

   { Test auf Element in Menge

3. **select** $A_1, A_2, \ldots, A_n$
   from $r_1, r_2, \ldots, r_m$
   where $\{r_i.A_k \mid (r_i.A_k, r_j.A_l, \ldots)\}$ {=|<>|<| ...} [any | all]
   (**select... from... where...**)

   { Test auf (Un)gleichheit

Bemerkungen:

- Geschachtelte Anfragen heißen auch Unterabfragen oder Subqueries. Subqueries erhöhen nicht die Ausdruckskraft von SQL, erleichtern jedoch die Formulierung von Anfragen. Die Semantik jeder Subquery lässt sich mit Join-Operationen nachbilden.

- Subqueries in der From-Klausel (*Derived Table Subqueries*) dienen zur Bildung spezialisierter Tabellen für kartesische Produkte. [Beispiel]

- Subqueries in der Where-Klausel (*Expression Subqueries*) dienen zur Formulierung von Bedingungen. Wichtige Verwendungsformen:

  1. Das Ergebnis der Subquery wird daraufhin getestet, ob es die leere Menge ist, d.h., ob es keinen oder ob es mindestens einen Match gibt. [Beispiel]

  2. Das Ergebnis der Subquery wird daraufhin getestet, ob es einen bestimmten Attributwert oder ein bestimmtes Tupel enthält.

  3. Ohne `any` bzw. `all`. Das Ergebnis der Subquery muss genau *ein* Element zurückliefern, das dann bzgl. der angegebenen Relation ($=, <>, <, \ldots$) getestet wird.

     Mit `any` bzw. `all`. Das Ergebnis der Subquery kann eine Menge sein.

     `=any` ist semantisch äquivalent zu `in`,
     `<>all` ist semantisch äquivalent zu `not in`.

Bemerkungen: (Fortsetzung)

- Alternativ zu „<>“ kann der Ungleich-Operator auch entsprechend als „!=“ notiert werden.
- Das Schlüsselwort `all` in der Where-Klausel bezeichnet keinen Allquantor. Mit `all` lässt sich nur *ein* Element einer Menge A mit den (allen) Elementen einer Menge B vergleichen. Ein Allquantor hingegen würde für *alle Elemente der Menge* A eine Bedingung formulieren.
- Subqueries können weiter geschachtelt werden, also ihrerseits Subqueries enthalten.
- In Subqueries kann auf Relationen der sie umschließenden Umgebung Bezug genommen werden. Stichwort: korrelierte Unterabfragen (*Correlated Subqueries*)
- Abhängig von der Strategie bzw. Güte der Anfrageoptimierung des DBMS können semantisch äquivalente Anfragen zu deutlich unterschiedlichen Antwortzeiten führen.

SQL als Datenanfragesprache

Geschachtelte Anfragen (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Anfrage

„Liefere die Kurs- und Angebotsnummern der Teilnehmer aus Bremen.“

SQL als Datenanfragesprache

Geschachtelte Anfragen (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Anfrage

„Liefere die Kurs- und Angebotsnummern der Teilnehmer aus Bremen.“

Relationenalgebra

$$\pi_{\text{KursNr,AngebotsNr}}(\text{nimmt\_teil} \bowtie \sigma_{\text{Ort}='\text{Bremen}'}(\text{Teilnehmer}))$$

SQL als Datenanfragesprache

Geschachtelte Anfragen (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Anfrage

„Liefere die Kurs- und Angebotsnummern der Teilnehmer aus Bremen.“

Relationenalgebra

$$\pi_{\text{KursNr,AngebotsNr}}(\text{nimmt\_teil} \bowtie \sigma_{\text{Ort}='\text{Bremen}'}(\text{Teilnehmer}))$$

SQL Variante (a)

```
select distinct nt.KursNr, nt.AngebotsNr
from nimmt_teil nt, Teilnehmer t
where nt.TeilnNr = t.TeilnNr and t.Ort = 'Bremen'
```

[SQLPad]

SQL als Datenanfragesprache

Geschachtelte Anfragen (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Anfrage

„Liefere die Kurs- und Angebotsnummern der Teilnehmer aus Bremen.“

Relationenalgebra

$$\pi_{\text{KursNr,AngebotsNr}}(\text{nimmt\_teil} \bowtie \sigma_{\text{Ort}='\text{Bremen}'}(\text{Teilnehmer}))$$

SQL Variante (b), korrelierte Unterabfrage

```
select distinct nt.KursNr, nt.AngebotsNr   [SQLPad]
from nimmt_teil nt
where exists   [Subquery-Verwendungsform 1]
       (select *
        from Teilnehmer t
        where t.Ort = 'Bremen' and t.TeilnNr = nt.TeilnNr)
```

SQL als Datenanfragesprache

Geschachtelte Anfragen (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Anfrage

„Liefere die Kurs- und Angebotsnummern der Teilnehmer aus Bremen.“

Relationenalgebra

$\pi_{\text{KursNr,AngebotsNr}}(\text{nimmt\_teil} \bowtie \sigma_{\text{Ort}='\text{Bremen}'}(\text{Teilnehmer}))$

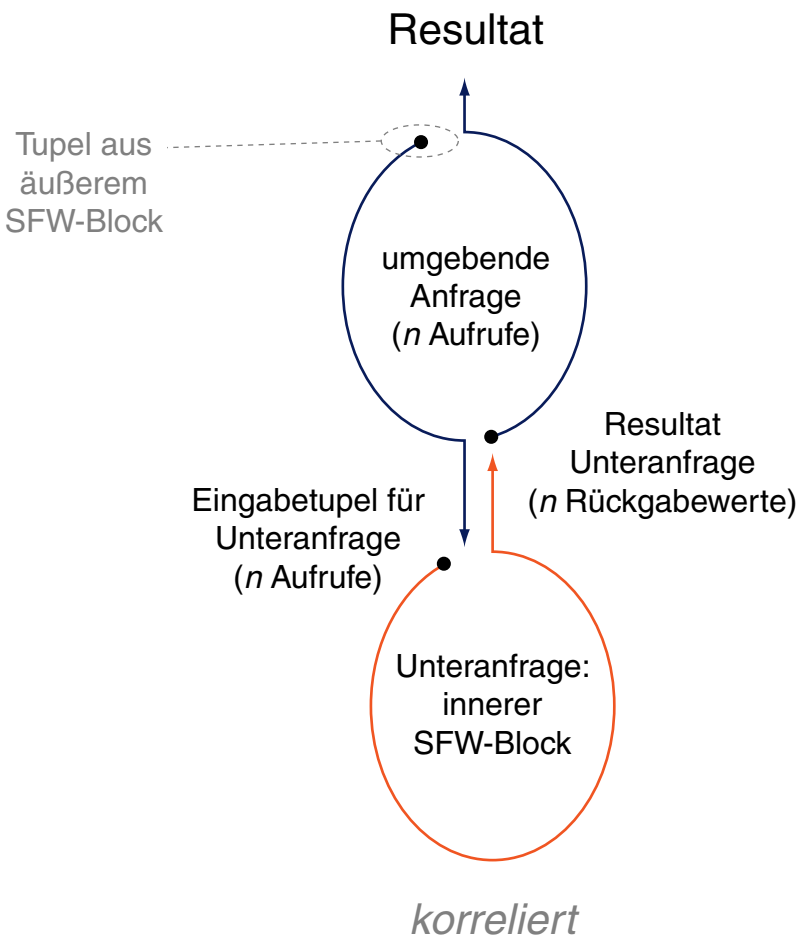
SQL Variante (c), unkorrelierte Unterabfrage

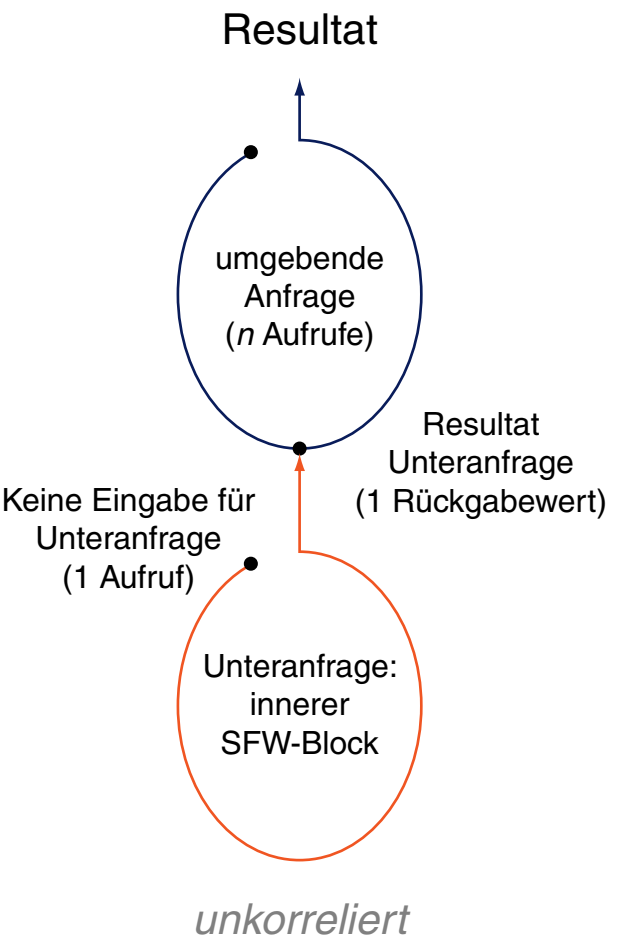
```
select distinct nt.KursNr, nt.AngebotsNr
from nimmt_teil nt
where nt.TeilnNr in
      (select TeilnNr
       from Teilnehmer
       where Ort = 'Bremen')
```

[SQLPad] [Subquery-Verwendungsform 2]

SQL als Datenanfragesprache

Geschachtelte Anfragen (Fortsetzung)





[vgl. Scholl, IS 2003]

SQL als Datenanfragesprache

Allquantifizierung

- SQL-89 und SQL-92 besitzen keinen Allquantor.
- Eine Allquantifizierung muss durch eine äquivalente Anfrage mit Existenzquantifizierung ausgedrückt werden.
- Alternativ kann eine Allquantifizierung auch mit der Aggregatfunktion `count` nachgebildet werden.

SQL als Datenanfragesprache

Allquantifizierung (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

Anfrage

„Wer nimmt an allen Kursen teil?“

SQL als Datenanfragesprache

Allquantifizierung (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

Anfrage

„Wer nimmt an allen Kursen teil?“

Relationenalgebra

$\pi_{\text{Name}}(\text{Teilnehmer} \bowtie (\text{nimmt\_teil} \div (\pi_{\text{KursNr}}(\text{Kurs}))))$

SQL als Datenanfragesprache

Allquantifizierung (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

Anfrage

„Wer nimmt an allen Kursen teil?“

Relationenalgebra

$\pi_{\text{Name}}(\text{Teilnehmer} \bowtie (\text{nimmt\_teil} \div (\pi_{\text{KursNr}}(\text{Kurs}))))$

Tupelkalkül

$$\{(t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge \forall t_3 \left(\neg\text{Kurs}(t_3) \vee \exists t_2 \, (\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr})\right)\}$$

SQL als Datenanfragesprache

Allquantifizierung (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

Anfrage

„Wer nimmt an allen Kursen teil?“

Relationenalgebra

$\pi_{\text{Name}}(\text{Teilnehmer} \bowtie (\text{nimmt\_teil} \div (\pi_{\text{KursNr}}(\text{Kurs}))))$

Tupelkalkül

$$\{(t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge \forall t_3 \left(\neg\text{Kurs}(t_3) \vee \exists t_2\ (\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr})\right)\}$$

$$\approx \{(t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge \nexists t_3 \neg \left(\underbrace{\neg\text{Kurs}(t_3)}_{\alpha} \vee \underbrace{\exists t_2\ (\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr})}_{\beta}\right)\}$$

SQL als Datenanfragesprache

Allquantifizierung (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

Anfrage

„Wer nimmt an allen Kursen teil?“

Relationenalgebra

$\pi_{\text{Name}}(\text{Teilnehmer} \bowtie (\text{nimmt\_teil} \div (\pi_{\text{KursNr}}(\text{Kurs}))))$

Tupelkalkül

$$\{(t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge \forall t_3 \left(\neg\text{Kurs}(t_3) \vee \exists t_2\ (\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr})\right)\}$$

$$\approx \{(t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge \nexists t_3 \neg \left(\underbrace{\neg\text{Kurs}(t_3)}_{\alpha} \vee \underbrace{\exists t_2\ (\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr})}_{\beta}\right)\}$$

$$\approx \{(t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge \nexists t_3 \left(\text{Kurs}(t_3) \wedge \nexists\ t_2\ (\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr})\right)\}$$

SQL als Datenanfragesprache

Allquantifizierung (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

Anfrage

„Wer nimmt an allen Kursen teil?“

Tupelkalkül

$$\{ (t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge \nexists t_3(\text{Kurs}(t_3) \wedge \nexists t_2(\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr})) \}$$

SQL [SQLPad]

```
select Name
from Teilnehmer t1
where not exists   [Subquery-Verwendungsform 1]
      (select *
       from Kurs t3
       where not exists
             (select *
              from nimmt_teil t2
              where t2.KursNr = t3.KursNr and t2.TeilnNr = t1.TeilnNr))
```

SQL als Datenanfragesprache

Allquantifizierung (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

Anfrage

„Wer nimmt an allen Kursen teil?“

Tupelkalkül

$$\{ (t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge \nexists t_3(\text{Kurs}(t_3) \wedge \nexists t_2(\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr})) \}$$

SQL [SQLPad]

```
select Name
from Teilnehmer t1
where not exists   [Subquery-Verwendungsform 1]
      (select *
       from Kurs t3
       where not exists
             (select *
              from nimmt_teil t2
              where t2.KursNr = t3.KursNr and t2.TeilnNr = t1.TeilnNr))
```

SQL als Datenanfragesprache

Allquantifizierung (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

Anfrage

„Wer nimmt an allen Kursen teil?“

Tupelkalkül

$\{ (t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge$
$\quad \nexists t_3(\text{Kurs}(t_3) \wedge \nexists t_2(\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr})) \}$

SQL [SQLPad]

```
select Name
from Teilnehmer t1
where not exists    [Subquery-Verwendungsform 1]
       (select *
        from Kurs t3
        where not exists
              (select *
               from nimmt_teil t2
               where t2.KursNr = t3.KursNr and t2.TeilnNr = t1.TeilnNr))
```

SQL als Datenanfragesprache

Allquantifizierung (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

Anfrage

„Wer nimmt an allen Kursen teil?“

Tupelkalkül

$\{ (t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge$
$\quad \nexists t_3(\text{Kurs}(t_3) \wedge \nexists t_2(\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr})) \}$

SQL [SQLPad]

```
select Name
from Teilnehmer t1
where not exists    [Subquery-Verwendungsform 1]
       (select *
        from Kurs t3
        where not exists
              (select *
               from nimmt_teil t2
               where t2.KursNr = t3.KursNr and t2.TeilnNr = t1.TeilnNr))
```

SQL als Datenanfragesprache

Allquantifizierung (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

Anfrage

„Wer nimmt an allen Kursen teil?“

Tupelkalkül

$$\{ (t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge \nexists t_3(\text{Kurs}(t_3) \wedge \nexists t_2(\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr})) \}$$

SQL [SQLPad]

```
select Name
from Teilnehmer t1
where not exists    [Subquery-Verwendungsform 1]
       (select *
        from Kurs t3
        where not exists
              (select *
               from nimmt_teil t2
               where t2.KursNr = t3.KursNr and t2.TeilnNr = t1.TeilnNr))
```

SQL als Datenanfragesprache

Allquantifizierung (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

Anfrage

„Wer nimmt an allen Kursen teil?“

Tupelkalkül

$\{ (t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge$
$\quad \nexists t_3(\text{Kurs}(t_3) \wedge \nexists t_2(\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr})) \}$

SQL [SQLPad]

```
select Name
from Teilnehmer t1
where not exists    [Subquery-Verwendungsform 1]
       (select *
        from Kurs t3
        where not exists
              (select *
               from nimmt_teil t2
               where t2.KursNr = t3.KursNr and t2.TeilnNr = t1.TeilnNr))
```

SQL als Datenanfragesprache

Allquantifizierung (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kurs	
KursNr	Titel
G08	Graphentheorie
P13	Datenbanken
G10	Modellierung

Anfrage

„Wer nimmt an allen Kursen teil?“

Tupelkalkül

$$\{ (t_1.\text{Name}) \mid \text{Teilnehmer}(t_1) \wedge \nexists t_3(\text{Kurs}(t_3) \wedge \nexists t_2(\text{nimmt\_teil}(t_2) \wedge t_2.\text{KursNr} = t_3.\text{KursNr} \wedge t_2.\text{TeilnNr} = t_1.\text{TeilnNr})) \}$$

SQL [SQLPad]

```
select Name
from Teilnehmer t1
where not exists    [Subquery-Verwendungsform 1]
       (select *
        from Kurs t3
        where not exists
              (select *
               from nimmt_teil t2
               where t2.KursNr = t3.KursNr and t2.TeilnNr = t1.TeilnNr))
```

Bemerkungen:

- Natürlichsprachliche Formulierung der SQL-Anfrage:
  „Liefere jeden Teilnehmer, bei dem kein Kurs existiert, an dem er nicht teilnimmt.“
- Siehe auch: “Relational Division in SQL The Easy Way.” [gregorulm.com]

Bemerkungen: (Fortsetzung)

- Wiederholung. Bei (formalen, logischen, natürlichen) Sprachen unterscheidet man zwischen Sätzen aus der Sprache selbst und der Formulierung von Zusammenhängen *über* solche Sätze. Sätze aus der Sprache selbst dienen uns zur Kommunikation mittels dieser Sprache; die Symbole, die vewendet werden, um solche Sätze zu formulieren, gehören zur Objektsprache. Symbole, die verwendet werden, um *über* Sätze zu sprechen, die in der Objektsprache formuliert sind, gehören zur Metasprache. [DB:V Formelsemantik]

- Wiederholung. Die Formelbezeichner α, β, γ, die Prädikatsbezeichner P, Q, die Quantoren $\forall$, $\exists$, die Variablenbezeichner t, x, y, z, und die Junktoren $\neg$, $\wedge$, $\vee$, $\rightarrow$ gehören zur Objektsprache. Das $\approx$-Zeichen ist ein Zeichen der Metasprache und steht für „ist logisch äquivalent mit“.

  Es gelten u.a. folgende Äquivalenzen: [DB:V Formelsemantik]

$$\begin{aligned} \neg(\alpha \vee \beta) &\approx \neg\alpha \wedge \neg\beta && \text{(deMorgan)} \\ \neg(\alpha \wedge \beta) &\approx \neg\alpha \vee \neg\beta \end{aligned}$$

$$\begin{aligned} \alpha \rightarrow \beta &\approx \neg\alpha \vee \beta && \text{(Implikation)} \\ (\alpha \wedge \beta) \rightarrow \gamma &\approx \neg\alpha \vee \neg\beta \vee \gamma \end{aligned}$$

$$\begin{aligned} \forall x P(x) &\approx \neg\exists x(\neg P(x)) && \text{(Quantoren)} \\ \forall x(\neg P(x)) &\approx \neg\exists x P(x) \end{aligned}$$

- Quantoren binden so stark wie $\neg$.

SQL als Datenanfragesprache

Mengenoperationen

Seien r_1, r_2 Relationen über den Schemata $\mathcal{R}_1 = \{A_1, A_2\}$ bzw. $\mathcal{R}_2 = \{A_2, A_3\}$. Mengenoperationen sind nur für „kompatible“ Attributlisten erlaubt.

- Vereinigung.

```
select A2 from r1 where <condition1>
union [all]
select A2 from r2 where <condition2>
```

SQL als Datenanfragesprache

Mengenoperationen

Seien r_1, r_2 Relationen über den Schemata $\mathcal{R}_1 = \{A_1, A_2\}$ bzw. $\mathcal{R}_2 = \{A_2, A_3\}$. Mengenoperationen sind nur für „kompatible“ Attributlisten erlaubt.

- Vereinigung.

```
select A2 from r1 where <condition1>
union [all]
select A2 from r2 where <condition2>
```

- Durchschnitt.

```
select A2 from r1, r2
where r1.A2 = r2.A2 and <condition1> and <condition2>
```

SQL als Datenanfragesprache

Mengenoperationen

Seien r_1, r_2 Relationen über den Schemata $\mathcal{R}_1 = \{A_1, A_2\}$ bzw. $\mathcal{R}_2 = \{A_2, A_3\}$. Mengenoperationen sind nur für „kompatible“ Attributlisten erlaubt.

- Vereinigung.

```
select A2 from r1 where <condition1>
union [all]
select A2 from r2 where <condition2>
```

- Durchschnitt.

```
select A2 from r1, r2
where r1.A2 = r2.A2 and <condition1> and <condition2>
```

- Differenz.

```
select A2 from r1
where <condition1> and r1.A2 not in    [Subquery-Verwendungsform 2]
      (select A2 from r2 where <condition2>)
```

Bemerkungen:

- In der Relationenalgebra sind Mengenoperationen nur über Relationen mit gleichen Relationenschemata zugelassen.
- In SQL spielen im Zusammenhang mit Mengenoperationen die Namen der Attribute keine Rolle: Mengenoperationen erfordern nur, dass die Listen der Attribute der beteiligten Relationen positionsweise *kompatibel* sind.
- Zwei Attribute sind kompatibel zueinander, falls sie kompatible Wertebereiche haben. Das heißt, dass die Wertebereiche entweder
  1. gleich sind oder
  2. beide auf dem Typ „Character“ basieren oder
  3. beide von einem numerischen Typ sind.
- `union` eliminiert Duplikate,
  `union all` bildet eine Multimenge.

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen

Aggregatfunktionen, auch Built-in-Funktionen genannt, führen Operationen auf Tupel*mengen* durch und „verdichten“ so eine Menge von Werten zu einem einzelnen Wert.

Aggregatfunktionen in SQL-89:

Name	Semantik
`count(*)`	Anzahl der Tupel
	=
`count(<attribute>)`	Anzahl der Attributausprägungen
`count(distinct <attribute>)`	Anzahl verschiedener Attributausprägungen
`max(<attribute>)`	Maximum der Attributausprägungen
`min(<attribute>)`	Minimum der Attributausprägungen
`avg([distinct] <attribute>)`	Durchschnitt der Attributausprägungen
`sum([distinct] <attribute>)`	Summe der Attributausprägungen

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen

Aggregatfunktionen, auch Built-in-Funktionen genannt, führen Operationen auf Tupel*mengen* durch und „verdichten“ so eine Menge von Werten zu einem einzelnen Wert.

Aggregatfunktionen in SQL-89:

Name	Semantik
`count(*)`	Anzahl der Tupel
	=
`count(<attribute>)`	Anzahl der Attributausprägungen
`count(distinct <attribute>)`	Anzahl verschiedener Attributausprägungen
`max(<attribute>)`	Maximum der Attributausprägungen
`min(<attribute>)`	Minimum der Attributausprägungen
`avg([distinct] <attribute>)`	Durchschnitt der Attributausprägungen
`sum([distinct] <attribute>)`	Summe der Attributausprägungen

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kursleiter		
PersNr	Name	PersAlter
11231	Suermann	39
21672	Lettmann	46
31821	Curatolo	51

Anfragen [SQLPad]

„Liefere die Anzahl aller Kursteilnehmer.“

```
select count(*)
from Teilnehmer
```

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kursleiter		
PersNr	Name	PersAlter
11231	Suermann	39
21672	Lettmann	46
31821	Curatolo	51

Anfragen [SQLPad]

„Liefere die Anzahl aller Kursteilnehmer.“

```
select count(*)
from Teilnehmer
```

„Wieviele Teilnehmer kommen aus Hamburg?“

```
select count(*)
from Teilnehmer
where Ort = 'Hamburg'
```

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kursleiter		
PersNr	Name	PersAlter
11231	Suermann	39
21672	Lettmann	46
31821	Curatolo	51

Anfragen [SQLPad]

„Liefere die Anzahl aller Kursteilnehmer.“

```
select count(*)
from Teilnehmer
```

„Wieviele Teilnehmer kommen aus Hamburg?“

```
select count(*)
from Teilnehmer
where Ort = 'Hamburg'
```

„Wie ist das Durchschnittsalter der Kursleiter?“

```
select avg(PersAlter)
from Kursleiter
```

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kursleiter		
PersNr	Name	PersAlter
11231	Suermann	39
21672	Lettmann	46
31821	Curatolo	51

Anfragen [SQLPad]

„Wie ist die Personalnummer des ältesten Kursleiters?“

```
select
from
where
```

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kursleiter		
PersNr	Name	PersAlter
11231	Suermann	39
21672	Lettmann	46
31821	Curatolo	51

Anfragen [SQLPad]

„Wie ist die Personalnummer des ältesten Kursleiters?“

```
select PersNr
from Kursleiter
where PersAlter =    [Subquery-Verwendungsform 3]
       (select max(PersAlter)
        from Kursleiter)
```

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kursleiter		
PersNr	Name	PersAlter
11231	Suermann	39
21672	Lettmann	46
31821	Curatolo	51

Anfragen [SQLPad]

„Wie ist die Personalnummer des ältesten Kursleiters?“

```
select PersNr
from Kursleiter
where PersAlter =    [Subquery-Verwendungsform 3]
       (select max(PersAlter)
        from Kursleiter)
```

„Liefere die Kurs- und Angebotsnummern der Teilnehmer aus Bremen.“ [andere Varianten]

```
select nt.KursNr, nt.AngebotsNr
from nimmt_teil nt
where 0 <            [Subquery-Verwendungsform 3]
       (select count(*)
        from Teilnehmer t
        where t.Ort = 'Bremen' and t.TeilnNr = nt.TeilnNr)
```

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen (Fortsetzung)

Teilnehmer		
TeilnNr	Name	Ort
143	Schmidt	Bremen
145	Huber	Augsburg
146	Abele	Bochum

nimmt_teil		
AngebotsNr	KursNr	TeilnNr
2	G08	143
2	P13	143
1	G08	145

Kursleiter		
PersNr	Name	PersAlter
11231	Suermann	39
21672	Lettmann	46
31821	Curatolo	51

Anfragen [SQLPad]

„Wie ist die Personalnummer des ältesten Kursleiters?“

```
select PersNr
from Kursleiter
where PersAlter =    [Subquery-Verwendungsform 3]
       (select max(PersAlter)
        from Kursleiter)
```

„Liefere die Kurs- und Angebotsnummern der Teilnehmer aus Bremen.“ [andere Varianten]

```
select nt.KursNr, nt.AngebotsNr
from nimmt_teil nt
where 0 < exists    [Subquery-Verwendungsform 1]
       (select count(*) *
        from Teilnehmer t
        where t.Ort = 'Bremen' and t.TeilnNr = nt.TeilnNr)
```

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen (Fortsetzung)

```
 select ...
 from ...
[where ...]
[group by <attribute1>, <attribute2>, ...]
[having <condition>]
```

- Die Group-by-Klausel bewirkt eine Gruppen- bzw. Teilmengenbildung: Alle Tupel, die gleiche Werte in der spezifizierten Attributliste haben, bilden eine Teilmenge.
- Aggregatfunktionen werden auf die Teilmengen angewandt.
- Jede Teilmenge wird durch *ein* Tupel repräsentiert. Folglich können in der Select-Klausel nur Aggregatfunktionen oder Attribute, nach denen gruppiert wird, vorkommen, damit nur *ein* Tupel resultiert.
- Mittels einer Having-Klausel lassen sich die Teilmengen (genauer: die sie repräsentierenden Tupel) weiter filtern.

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen (Fortsetzung)

```
 select ...
 from ...
[where ...]

[group by <attribute1>, <attribute2>, ...]
[having <condition>]
```

- Die Group-by-Klausel bewirkt eine Gruppen- bzw. Teilmengenbildung: Alle Tupel, die gleiche Werte in der spezifizierten Attributliste haben, bilden eine Teilmenge.
- Aggregatfunktionen werden auf die Teilmengen angewandt.
- Jede Teilmenge wird durch *ein* Tupel repräsentiert. Folglich können in der Select-Klausel nur Aggregatfunktionen oder Attribute, nach denen gruppiert wird, vorkommen, damit nur *ein* Tupel resultiert.
- Mittels einer Having-Klausel lassen sich die Teilmengen (genauer: die sie repräsentierenden Tupel) weiter filtern.

Bemerkungen:

- Logische Ausführungsreihenfolge: from $\rightarrow$ where $\rightarrow$ group by $\rightarrow$ having $\rightarrow$ select
- Unterschied zwischen der Where-Klausel und der Having-Klausel: Mit der Where-Klausel werden Tupel gefiltert, mit der Having-Klausel werden Tupel*mengen* gefiltert.
- Aggregatfunktionen können nicht geschachtelt werden.

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen (Fortsetzung)

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

MitarbeiterSkill	
PersNr	Skill
1234	Java
1234	C++
3334	Linux
3334	Java
9998	Linux
9876	DB2

Anfrage

„Für welche Programmierfähigkeiten gibt es mehrere Mitarbeiter?“

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen (Fortsetzung)

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

MitarbeiterSkill	
PersNr	Skill
1234	Java
1234	C++
3334	Linux
3334	Java
9998	Linux
9876	DB2

Anfrage

„Für welche Programmierfähigkeiten gibt es mehrere Mitarbeiter?“

```
select count(*) as Anzahl, Skill as Faehigkeit
from MitarbeiterSkill
group by Skill
having Anzahl > 1
```

[SQLPad]

$\rightsquigarrow$

Anzahl	Faehigkeit
2	Java
2	Linux

SQL als Datenanfragesprache

Aggregat- und Gruppierungsfunktionen (Fortsetzung)

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

MitarbeiterSkill	
PersNr	Skill
1234	Java
1234	C++
3334	Linux
3334	Java
9998	Linux
9876	DB2

Anfrage

„Wie ist die Verteilung der Fähigkeiten in Abteilung 5?“

```
select count(*) as Anzahl, Skill as Faehigkeit
from Mitarbeiter m, MitarbeiterSkill s
where AbtNr = 5 and m.PersNr = s.PersNr
group by Skill
```

[SQLPad]

$\rightsquigarrow$

Anzahl	Faehigkeit
2	Java
1	C++
1	Linux

SQL als Datenanfragesprache

Nullwerte

Der spezielle Wert „Null“, $\perp$, ist als Wert in jedem Datentyp vorhanden. Gründe, die zur Verwendung von Null als Attributausprägung führen:

1. Der Wert des Attributes ist unbekannt.
2. Der Wert des Attributes ist bekannt, soll aber nicht gespeichert werden.
3. Im Wertebereich des Attributes ist kein adäquater Wert vorhanden.

SQL als Datenanfragesprache

Nullwerte

Der spezielle Wert „Null“, $\perp$, ist als Wert in jedem Datentyp vorhanden. Gründe, die zur Verwendung von Null als Attributausprägung führen:

1. Der Wert des Attributes ist unbekannt.
2. Der Wert des Attributes ist bekannt, soll aber nicht gespeichert werden.
3. Im Wertebereich des Attributes ist kein adäquater Wert vorhanden.

Verarbeitung von Null-Werten:

(a) Eine arithmetische Operation ergibt Null, falls *ein* Operand Null ist.

(b) Für den Test auf Null dienen die Operatoren `is` bzw. `is not`.

[SQLPad]	Beispiele	Wert
zu a)	`4+null, 4<null, null=null`	`null`
zu b)	`null is null`	`true`
	`null is not null`	`false`

SQL als Datenanfragesprache

Nullwerte (Fortsetzung)

Mit Hilfe von Null ist in SQL eine dreiwertige Logik realisiert.

Boolesche Semantik:

$\neg$	
true	false
unknown	unknown
false	true

SQL-Entsprechung:

not	
1	0
null	null
0	1

SQL als Datenanfragesprache

Nullwerte (Fortsetzung)

Mit Hilfe von Null ist in SQL eine dreiwertige Logik realisiert.

Boolesche Semantik:

$\neg$	
true	false
unknown	unknown
false	true

$\wedge$	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

SQL-Entsprechung:

not	
1	0
null	null
0	1

and	1	null	0
1	1	null	0
null	null	null	0
0	0	0	0

SQL als Datenanfragesprache

Nullwerte (Fortsetzung)

Mit Hilfe von Null ist in SQL eine dreiwertige Logik realisiert.

Boolesche Semantik:

$\neg$	
true	false
unknown	unknown
false	true

$\wedge$	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

$\vee$	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

SQL-Entsprechung:

not	
1	0
null	null
0	1

and	1	null	0
1	1	null	0
null	null	null	0
0	0	0	0

or	1	null	0
1	1	1	1
null	1	null	null
0	1	null	0

Bemerkungen:

- In einer Where-Klausel werden nur Tupel berücksichtigt, die zu `true` evaluieren. Das heißt, Tupel, die zu `unknown` evaluieren, werden nicht in das Ergebnis aufgenommen.
- Bei einer Gruppierung wird `null` als eigenständiger Wert interpretiert und in einer eigenen Gruppe zusammengefasst.

SQL als Datenanfragesprache

Zusammengesetzte Terme

In Select- und Where-Klauseln können an der Stelle von Attributen auch zusammengesetzte Terme stehen, wie z.B. arithmetische Ausdrücke oder Ausdrücke, die Zeichenketten manipulieren.

Kursleiter		
PersNr	Name	PersAlter
11231	Suermann	39
21672	Lettmann	46
31821	Curatolo	51

```
select PersNr*PersAlter as Glueckszahl
from Kursleiter
```
[SQLPad]

$\rightsquigarrow$

Glueckszahl
438009
996912
1622871

SQL als Datenanfragesprache

SQL-89 versus SQL-92: Joins

- kartesisches Produkt und Equi-Join in SQL-89:

```
select *
from Mitarbeiter, MitarbeiterSkill

select *
from Mitarbeiter, MitarbeiterSkill
where Mitarbeiter.PersNr = MitarbeiterSkill.PersNr
```

SQL als Datenanfragesprache

SQL-89 versus SQL-92: Joins

- kartesisches Produkt und Equi-Join in SQL-89:

```
select *
from Mitarbeiter, MitarbeiterSkill

select *
from Mitarbeiter, MitarbeiterSkill
where Mitarbeiter.PersNr = MitarbeiterSkill.PersNr
```

- kartesisches Produkt und Equi-Join in SQL-92:

```
select *
from Mitarbeiter cross join MitarbeiterSkill

select *
from Mitarbeiter join MitarbeiterSkill
    on Mitarbeiter.PersNr = MitarbeiterSkill.PersNr
```

SQL als Datenanfragesprache

SQL-89 versus SQL-92: Joins (Fortsetzung)

- natürlicher Verbund in SQL-92:

```
select *
from Mitarbeiter natural join MitarbeiterSkill

select *
from Mitarbeiter join MitarbeiterSkill
    using (PersNr)
```

SQL als Datenanfragesprache

SQL-89 versus SQL-92: Joins (Fortsetzung)

- natürlicher Verbund in SQL-92:

```
select *
from Mitarbeiter natural join MitarbeiterSkill

select *
from Mitarbeiter join MitarbeiterSkill
    using (PersNr)
```

- äußere Verbunde in SQL-92:

```
[natural] {left | right} outer join
    using (...)
    on ...
```

SQL als Datenanfragesprache

SQL-89 versus SQL-92: Joins (Fortsetzung)

Mitarbeiter				
Name	PersNr	Wohnort	ChefPersNr	AbtNr
Smith	1234	Weimar	3334	5
Wong	3334	Koeln	8886	5
Zelaya	9998	Erfurt	9876	4
Wallace	9876	Berlin	8886	4

MitarbeiterSkill	
PersNr	Skill
1234	Java
1234	C++
3334	Linux
3334	Java
9998	Linux
9876	DB2

Anfrage

„Wie sind die Personalnummern der Chefs, die keine Ahnung haben?“

```
select distinct ChefPersNr  [SQLPad]
from Mitarbeiter m left outer join Mitarbeiterskill s
   on m.ChefPersNr = s.PersNr
where Skill is null
```

$\rightsquigarrow$

ChefPersNr
8886

SQL als Datenanfragesprache

SQL-89 versus SQL-92

- Es gibt die Mengenoperationen `union`, `intersect` und `except`, für die mittels `corresponding` noch eine Attributliste vereinbart werden kann.
- In der For-Klausel können mittels eines Select-From-Where-Blocks abgeleitete Relationen erzeugt und benannt werden.
- In Bedingungen der Where-Klausel lassen sich nicht nur skalare Ausdrücke, sondern auch Tupel spezifizieren.